Honors Compilers

An Introduction to Algol-68S

Jan 24th 2002
Closed-Clause (Block)

- Closed-clause is basic unit of language
  - **begin** declarations and statements **end**
    - begin/end can be replaced by ()
    - No termination semicolon
    - Semicolon is a separator NOT a terminator
    - Declarations and statements can be freely mixed
    - Begin might be written BEGIN or .begin
    - All constructs have values (possibly void value)
Predefined Modes

- A Mode in A68S is what other languages call a type.
- Predefined modes are
  - `int`
  - `real`
  - `bool`
  - `Char`
- Note that mode names are always bold
Declarations

- General form is
  - mode name = value
  - This is a constant declaration, the value of the name cannot be changed, it is bound to the given value for the remainder of the block.

- Example
  - **real** pi = 3.14159;
Reference modes (variables)

- The mode:
  - `ref` `amode`
  - Where `amode` is any mode, e.g. `int` is a mode that represents a value reference-to-amode
  - You can think of this as a pointer
  - Or a variable (A68 unifies the notions of variable and pointer)
Now we can declare a variable

- To declare an integer variable:
  - `ref int` `ivar = ?`
  - But what shall we use for `?`
  - We need an expression that returns a value of mode `ref int`
- Two possibilities:
  - `loc int` (local integer allocated on stack frame)
  - `heap int` (integer allocated on GC heap)
Declaring a Variable

- So now we can write the complete declaration of an integer variable
  - `ref int ivar = loc int`
- That’s a bit verbose, so we allow a simpler form that means exactly the same:
  - `int ivar`
- But you need to remember the long form
Assignment Statements

- Assignment is represented by the symbol := which takes two values.
- The value on the left must be of mode ref amode where amode is some mode. This is also the result of the assignment construct.
- The value on the right side must be of mode amode.
- For example:
  - ivar := 13;
  - Ivar itself is not changed, but the int which it references is changed.
Casts

Suppose we have two variables:

- `int` `ivar`; `int` `jvar`
- Can we write: `ivar := jvar`
- Not according to rules so far, since `jvar` is not of mode `int`, but of mode `ref int`.
- But if you have a `ref int`, how can you get an `int`? You can “cast” to `int`, i.e. get the value by dereferencing the pointer: `ivar := (int) jvar;`
Dereferencing Coercion

- Consider again the previous assignment
  \( \text{ivar} := (\text{int}) \ j\text{var}; \)

- It would be painful to require the explicit cast on every such assignment, so we can leave it out:
  \( \text{ivar} := \text{jvar}; \)

- One level of \textbf{ref} can be removed from the right side automatically if necessary
The Widening Coercion

- Suppose we have
  ```
  int Jvar;
  float Fvar;
  ```
- We can widen an int to float with a cast
  ```
  Fvar := (float) Jvar;
  ```
- But this coercion is also allowed implicitly
  ```
  Fvar := Jvar;
  ```
Operators

- Operators like + (addition) can be used to form expression values in a conventional manner:
  
  \[ 3 + 4 \]

- This is the addition operator that takes two int values, and yields an int value. Implicit dereferencing is permitted here:
  
  \[ \text{Ivar} + \text{Jvar} \]
Value of a block

- Statements and Declarations are separated by semicolons
- Formally, semicolon is an operator that “voids” (another coercion, convert to void) its left argument and returns the value of its right argument.
- Thus the value of a block is the value of the last statement.
- Ordinary parenthesization is special case
Control Structures, Conditionals

- **if** condition **then**
  statements
- **else**
  statements
- **fi**;

- Condition is of mode bool (dereferencing is allowed here)
- Coerced modes of both branches must be the same, value is value of appropriate statement sequence.
Control Structures, Conditionals

- **if** condition **then**
  statements

- **elif** condition **then**
  statements

- **elif** condition **then**
  statements

- **else**
  statements

- **fi;**
Control Structures, Loops

- **for** index
  - **from** first
  - **by** incr
  - **to** last
- **while** condition **do**
  - statements
- **od**

- Can leave out any phrase
- Value of loop is void
Case Statements

- **case** Ivar **in**
  
  expr, expr, expr, ... expr

  **esac**

- The expr here may be, for example

  begin ... **if** ... **fi**; .... **end**;
LHS condition (not new feature)

- Consider:

  ```
  if Ivar > Jvar then Ivar else Jvar fi
  := 3;
  ```

- Left side has value Ivar or Jvar, i.e. is of mode `int`, so quite suitable for assignment.
Multiple assign (not new feature)

- Consider:
  
  Ivar := Jvar := 3;

- Which means
  
  Ivar := (Jvar := 3);

- Right side has mode \texttt{ref int} (Jvar)

- But can be dereferenced to \texttt{int}

- So assignment to Ivar is fine
Renaming (not a new feature)

- Consider
  ```
  ref int Ivar = loc int := 2;
  ref int Jvar = Ivar;
  ...
  ```

  Jvar is a renaming of Ivar
Pointer processing (not new feature)

- Consider

  ```
  ref int Jvar = loc int;
  ref ref int Iptr = loc ref int;
  ...
  Iptr := Jvar;
  Jvar := 4;
  ```

  Iptr points to a ref int variable that points to an int that contains the value 4.

  ```
  (ref int) Iptr := 3;
  ```

  Jvar now contains 3
Complex modes

- Struct modes are like records in Pascal or structs in C
  ```c
  mode list = struct (int val, ref list next);
  ```

- Row modes are like arrays
  ```c
  mode vector = [10] int;
  mode array = [1:N, 3:M] float;
  ```
Union modes

- Free unions (no explicit tag)

```c
mode intreal = union (int, real);
ref intreal irvar = loc intreal;
irvar := 3;  (the uniting coercion)
irvar := ivar;  (dereferencing, then uniting)
```
Procedures

- proc intmax (int a,b) int:
  if a > b then a else b fi;
- All parameters passed by value, but we can pass parameters of type ref amode
- proc apply (ref [] real a, proc (real) real f):
  for I from lwb a to upb a do
    a[i] := f (a[I]);
  od